The knowledge package
[2017/10/10 v1.06]

Thomas Colcombet
thomas.colcombet@irif.fr

October 10, 2017

Abstract

The knowledge package offers automatic tools and commands for helping the writer of a (scientific) document to have all notions (hyper)linked to places where these are defined. Using the standard tools of \LaTeX for doing that would be extremely tedious. Eventually, it is intended to use the package for producing scientific semantic aware documents.

Status of this version

contact: thomas.colcombet@irif.fr
version: 2017/10/10 v1.06 (produced October 10, 2017)
license: \LaTeX Project Public License 1.2
web: https://www.irif.fr/~colcombe/knowledge_en.html
CTAN: https://www.ctan.org/pkg/knowledge

This is still in beta version but close to a first definitive release. A lot of functionalities are perfectly operational, and the syntax is close to stable. At any rate, if changes happen, these will only affect a few lines in the configuration of the document. Nevertheless, it is still advised to put the package directly in the document’s directory rather than in the search tree of \LaTeX.
# Contents

1 History 4  
2 Quick start 7  
   2.1 Linking to outer documents/urls, and to labels 7  
   2.2 Linking inside a document 9  
   2.3 Mathematics 11  
3 Usage of the knowledge package 12  
   3.1 Options and configuration 12  
      3.1.1 Options at package loading 12  
      3.1.2 Writing mode 12  
      3.1.3 Automatic loading of other packages 13  
      3.1.4 Configuring and \knowledgeconfigure 14  
   3.2 What is a knowledge? 14  
   3.3 The \knowledge command and variations 15  
      3.3.1 General description of the \knowledge command 15  
      3.3.2 Targeting and the corresponding directives 16  
      3.3.3 General directives 17  
      3.3.4 Knowledge styles and the \knowledgestyle command 18  
      3.3.5 New directives: the \knowledgedirective command 19  
      3.3.6 \knowledgestyle versus \knowledgedirective 19  
      3.3.7 Default directives: the \knowledgedefault command 20  
   3.4 The \kl command 20  
      3.4.1 The standard syntax 20  
      3.4.2 The quotation notation 21  
      3.4.3 Variants of \kl, \knowledgenewvariant, \knowledgevariantmodifier 22  
      3.4.4 Examples of variants of \kl 24  
   3.5 Scoping 26  
      3.5.1 What is the structure of scopes in a document 26  
      3.5.2 How is chosen the scope of a knowledge? 27  
      3.5.3 Accessing other scopes, the \knowledgeimport command 27  
      3.5.4 Managing scoping environments 28  
   3.6 Error handling 29  
   3.7 The diagnose file 29  
   3.8 Other packages 30  
      3.8.1 The xcolor option 30  
      3.8.2 The hyperref option 31  
      3.8.3 The makeidx option 35  
   3.9 Dealing with math 35  
   3.10 Fixes 35  
   3.11 Predefined configuration 36  
      3.11.1 The notion directive 36
4 Some questions and some answers

4.1 How to compile? ................................................................. 37
4.2 Problem with \item parameters ........................................... 37
4.3 Knowledges and moving arguments (table of contents, ...) .......... 37
4.4 Problems with tikzcd and other issues with the quotation notation 38
4.5 Problems with amsmath ....................................................... 39
4.6 Hyperref complains ........................................................... 39
4.7 Incorrect display ............................................................... 39
4.8 Editor ................................................................................. 40
  4.8.1 Emacs editor and quotes ................................................. 40
4.9 Others ................................................................................. 40

5 Resources ............................................................................ 41

5.1 List of commands ............................................................... 41
5.2 List of environments .......................................................... 41
5.3 List of directives (to use with \knowledge) .............................. 41
5.4 List of configuration directives (to use with \knowledgeconfigure) 42
1 History

2016-06-07 \knowledgemacro is now renamed to \knowledgedirective.
2017-01-13 \AP has been recoded, and is now more properly aligned in the margin. The visible anchor points option has also been made usable without the xcolor package.
2017-01-13 The package scope option can now be omitted. This in particular avoid clashes with the over-restriction on the structure of the document it entails. It should be improved to stop overloading the \begin command.
2017-01-14 The overloading of \begin and \end was done as protected commands, which should not be the case to be consistent with the behaviour of LaTeX (for instance, this was giving an extra line in the title in the conference mode of the class IEEEtran). Corrected: these commands are not protected anymore.
2017-01-15 A workaround for an incompatibility between the hyperref and the twocolumn mode as been added in the macro \knowledgeFixHyperrefTwocolumn (thanks to Daniela Petrisan).
2017-01-15 Added the directive synonym.
2017-01-15 Added the noknowledge package for minimizing the effects of not having knowledge activated.
2017-01-17 Changed the way options are handled, decoupling the package options (options of \usepackage) from the configuration options (see \knowledgeconfigure).
2017-01-17 Proper treatment of ‘final’ option and composition options.
2017-01-17 Added \IfKnowledgeFinalMode [TF] commands for the user.
2017-01-17 Added the option fix hyperref twocolumn as a shorthand for calling \knowledgeFixHyperrefTwocolumn (thanks to Daniela Petrisan and Luca Reggio).
2017-01-18 Added the configuration option notion that offers a basic configuration compatible with xcolor or not, and final and composition modes.
2017-01-19 Added \phantomintro and an explanation on how to deal with align*.
2017-02-20 Removed the warnings of latex for unknown labels in autoref.
2017-02-20 Removed nasty error making \AP not operative when anchor points were not visible.
2017-02-21 Added the protect link directive.
2017-02-21 Added the hyperlinks= configuration.
2017-02-27 visible anchor points is active by default now.
2017-02-27 A simple example is now included.
2017-02-28 Added the scope environment.
2017-02-28 Added the protect link and unprotect link configuration directives.
2017-02-28 Added the \knowledgeconfigureenvironment command.
2017-03-03 Added the breaklinks faq (thanks to Luca Reggio for the request).
2017-03-10 Added the “...” and “...@…” notations and the quotation mode (requested by Gabriele Puppis and Andreas Krebs).
2017-03-11 Added the “...@…” and “...@…” notations.
2017-03-13 Corrected for being compatible with version of expl3 posterior to Mars 2015 (\c_sys_jobname_str does not exist anymore). (Thanks to Jean-Éric Pin).
2017-03-14 Corrected that the @ letter was left a letter after \knowledgeFixHyperrefTwocolumn.
2017-04-09 Internal change of code, for scope handling and for the quotation notation: slowly going toward an extended quotation notation that can make the scope of search explicit.
2017-04-09 Added the `protect quotation` configure option, that is given a list of environments, and deactivates automatically the `quotation notation` when in there environments. This is a simple code for the moment. Typically, one can use `\knowledgeconfigure{protect quotation=tikzcd}`. For the moment, it is not explained in the document.

2017-04-19 Changed the display code such hat nested knowledges behave properly: before, the introduction would be performed for the object and the subobjects.

2017-04-20 The `electronic mode` has been added, and the `final mode` is now renamed into `paper mode`. The `\knowledgepackagemode` configuration variable is also available for easier scripting.

2017-06-06 FAQ on deactivating the quote in Emacs (thanks to Sylvain Perifel).

2017-06-08 Removed the `noknowledge` package and all references to it.

2017-06-08 Removed the `knowledgeutils.sty` and `scopearticle.sty` which are now integrated in the main file.

2017-06-08 The file `knowledge-example.sty` as been improved.

2017-06-09 First release of version 1.0 on CTAN.

2017-06-10 Corrected the `quotation notation` to make it expandable for avoiding problems in table of contents (the @ was not working).

2017-06-11 Corrected a bug linked to changes of expl3 on recent distributions (pointed by Murray Eisenberg)

2017-06-27 Overloaded labels now perform an expansion of the argument (this was causing problems with biblatex).

2017-06-28 Options `log-declarations` of xparse package removed (causing clash with other packages, as pointed by Juliusz Chroboczek).


2017-07-01 Corrected a conflict between the `scope` and `makeidx` option.

2017-07-03 Scoping becomes operational.

2017-07-04 The documentation for `notion` and `intro notion` are added (thanks to Fabian Reiter).

2017-07-09 The boolean `environment_bool` has been added to areas in order to not always trigger the scope from an environment. Now the environment `document` has this Boolean set to false, and it resolves a bug with the package `standalone` noticed by Fabian Reiter.

2017-07-20 Scoping becomes fully operational, with the parenthesis notation of `\kl` and `\intro`. The use of scope has been recoded. Now scope links reuse implicitly the key as a link. Documentation updated.

2017-07-26 File and line numbers added in the `kaux` file. Added the option `diagnose line` to deactivate it.

2017-07-26 Corrections to the documentation.

2017-07-27 Version 1.03.

2017-07-28 Corrected a bug of scoping in the context of synonyms. Added ctan for producing the ctan zip file.

2017-08-06 Now passes the compliance test `check-declarations` of expl3 (thanks to
The \texttt{hidelinks} option of \texttt{hyperref} is now always activated.

Ancient version of \texttt{xparse} does not have \texttt{\textbackslash NewExpandableDocumentCommand}. Corrected.

Version 1.05 on CTAN.

Bug in the implementation of \texttt{\textbackslash knowledgenewvariant} (that was invisible).

Found and corrected (thanks to Marc Zeitoun).

Version 1.06 on CTAN.
2 Quick start

The knowledge package offers several capabilities for handling colors, changing the display style, defining internal and external hyperlinks, producing an index, etc... All these possibilities arise from defining explicitly or implicitly knowledges associated to terms in plain english (or other languages).

We start by describing a certain number of problems/scenarii that a user may be confronted to, and show how to solve them. The hyperlinks in this document have been generated using the knowledge package. In the subsequent questions, a more detailed account of how the package works and can be parameterized is given.

There is also a file knowledge-example.tex that can be used as a starting point.

2.1 Linking to outer documents/urls, and to labels

The problem 1 I have a lot of external url's that I would like to [[very] often] have a link to, but I do not want to always type the full url. I do not want to remember weird labels/internal references/macro names either.

A solution is as follows. One first loads the knowledge package with option hyperref using either:

```latex
\usepackage[hyperref,quotation]{knowledge}
```
or:

```latex
\usepackage{hyperref}
\usepackage[quotation]{knowledge}
```

If you want to use the "···" notation.

Then, in the preamble (or in an external file), one uses commands of the form:

```latex
\knowledge{latex}{url={https://en.wikipedia.org/wiki/LaTeX}}
```

This configures the text 'latex' to be associated with the sole directive url, which means an hyperreference to this address.

Finally in the body of the paper, the sole extra command \kl (or the "-symbol if the quotation option is activated) is used, with as parameter a text. This text is searched for, and the directives attached to it (here the url), are used for formatting its printing. Hence:

This package has been written for use in \kl{latex}.

or, if the quotation option is activated,

1 This resembles a lot a macro so far. It nevertheless differs in that: (a) if not defined, it does not make the compilation fail as a macro would, and thus does not interfere with the writing process, (b) any text can be used and not only alphabetic letters as in \TeX, (c) you do not have to care about the space after, and (d) in fact the machinery for resolving the meaning of a knowledge is much more powerful than simple macro execution.
This package has been written for use in "latex". yields

This package has been written for use in \LaTeX.

Variation. But in fact, I would like \texttt{latex} to also be properly typeset \LaTeX, and in \texttt{gray}. This requires to load the package with the \texttt{xcolor} option (for being able to use colors, obviously), or by loading the package \texttt{xcolor} before, and then modify the \texttt{knowledge} command using extra directives:

```
\knowledge{latex}{url=https://en.wikipedia.org/wiki/LaTeX, text=\LaTeX, color=gray}
```

yields with the same text

This package has been written for use in \LaTeX.

The directives \texttt{text=} and \texttt{color=} have quite obvious meaning. Directives can also control the style using \texttt{emphasize}, \texttt{boldface}, \texttt{italic}, \texttt{typewriter} and so on. See Section 5.3 for a complete list of directives.

Variation (synonyms). It happens very often that there are several ways to name a notion, because of capitalized letters, conjugacy, grammar, or simply because it is not explicitly named in the text. There are two ways to resolve this issue. The first is to use the syntax

```
\kl[knowledge]{text} or "text@knowledge"
```

the result is that the text \texttt{‘text’} is displayed, but urls, colors, etc from \texttt{‘knowledge’} are used.

Another more systematic way to do it is to declare synonyms. This can be achieved using a \texttt{‘|’} separated list of texts in the \texttt{optional parameter} of the \texttt{knowledge} command, it is possible to add a list of ‘synonyms’, such as in:

```
\knowledge{latex}[LaTeX|Latex|LATEX]{url=http://en.wikipedia.org/wiki/LaTeX, text=\LaTeX, color=gray}
```

This is interesting for people’s name that can be displayed in various ways depending on the context. Hence

```
\knowledge{Donald Ervin Knuth}[Donald Knuth|Knuth]{url=https://fr.wikipedia.org/wiki/Donald_Knuth}
```

would allow

```
\kl{Knuth} as well as \kl{Donald Knuth},
or simply "Knuth" as well as "Donald Knuth" and so on
```

\textbf{Hint.} This is a shorthand for \texttt{a synonym} (or \texttt{link=}) directive. For instance, with the code

```
\knowledge{D. Knuth}{link=Knuth}
```

then \texttt{\kl{D. Knuth}} would also point to the same url.

```
\knowledge{D. Knuth}{synonym}
```

would also work if used just after the definition of \texttt{Knuth}.\"
to all point to the same web address. It is even more convenient to use it for nouns that are sometimes in plural form or at the beginning of a sentence. Hence:

```
knowledge{group}[groups|Groups|group morphism|group morphisms|Group morphisms]
{url=https://en.wikipedia.org/wiki/Group_(mathematics)}
```

makes it possible to use the notions in many contexts:

"Groups" form a category when equipped with "group morphisms".

### 2.2 Linking inside a document

**The problem** 2 I am writing a long scientific document with many notions tied together (typically, I have made all my best for clarifying but nevertheless it remains obscure, or it is a long survey involving many subfields, or a book, or a PhD thesis\(^2\)). I would like all the notions be linked inside the document for being able in one click, whenever something is used, to jump to its definition. However, I do not want it to be a hassle when writing.

A solution is as follows. First load the `knowledge` package in the preamble:

```
\usepackage[xcolor,hyperref,notion,quotation]{knowledge}
```

with suitable options: `hyperref` for links, `xcolor` for colors (if required, but always advised), `quotation` for using the quotation notation and `notion` for automatic configuration of the `notion` directive.

Then write the document using `\intro` (or ""\ldots"" if `quotation` is activated) when a notion is defined/introduced, and `\kl` (or ""\ldots"" if `quotation` is activated) when it is used. For instance:

```
\AP A \intro{semigroup} is an ordered pair $(S,\cdot)$ where $\cdot$ is an associative binary operator over $S$. 
\AP A \intro{monoid} $(M,\cdot,1)$ is a \kl{semigroup} $(M,\cdot)$ together with a neutral element $1$.
```

or when the quotation notation is activated:

```
\AP A ""semigroup"" is an ordered pair $(S,\cdot)$ where $\cdot$ is an associative binary operator over $S$. 
\AP A ""monoid"" $(M,\cdot,1)$ is a "semigroup" $(M,\cdot)$ together with a neutral element $1$.
```

This yields

Note that the `\AP` command is made visible thanks to a red corner.

---

\(^2\)Reviewers should appreciate...
A semigroup is an ordered pair \((S, \cdot)\) where \(\cdot\) is an associative binary operator over \(S\).

A monoid \((M, \cdot, 1)\) is a semigroup \((M, \cdot)\) together with a neutral element 1.

Undefined knowledges are in brown (it is an important feature that the compilation does not fail: undefined knowledges should not interfere with the writing of the document, which is the purpose of all this). One can now see the list of such problems in the file ‘filename.diagnose’. One can in particular find in the ‘Undefined knowledges’ section:

\begin{verbatim}
\knowledge{semigroup}{}
\knowledge{monoid}{}
\end{verbatim}

Which means that both ‘monoid’ and ‘semigroup’ are unknown knowledges.

To solve this, let us copy these two (or more) lines in the paper\(^3\), adding the notion directive (which is a configured version of the autoref directive, meaning essentially that you want to use the features of the \intro command), i.e., in the preamble:

\begin{verbatim}
\knowledge{semigroup}{notion}
\knowledge{monoid}{notion}
\end{verbatim}

The result is then (after two compilations):

A semigroup is an ordered pair \((S, \cdot)\) where \(\cdot\) is an associative binary operator over \(S\).

A monoid \((M, \cdot, 1)\) is a semigroup \((M, \cdot)\) together with a neutral element 1.

Clicking on ‘semigroup’ now jumps to the place it was introduced, and very precisely at the location of the red corner depicting the presence of the \AP-command. If now one adds the option electronic while loading the package, then the colors and the red corner disappear. When using the option paper, the links are still there, but all texts are in black.

It is very often the case that for plain english (or other languages) some terms can be used in several forms; verbs can be conjugated; nouns can be plural, and so on. So usually the lines added to the file look more like:

\begin{verbatim}
\knowledge{semigroup}{semigroups|Semigroups}{notion}
\knowledge{monoid}{monoid|Monoids}{notion}
\end{verbatim}

Now, using code like

"Monoids" and "semigroups" play the same role from now.

will properly be linked to the definition of a semigroup and a monoid.

\footnote{It is good practice to use a separate file, something like ‘paper-knowledge.tex’.

\begin{verbatim}
\knowledge{semigroup}{semigroups|Semigroups}{notion}
\knowledge{monoid}{monoid|Monoids}{notion}
\end{verbatim}

Hint. The directives synonym and link= can be also convenient.
2.3 Mathematics

The examples above show various techniques for using knowledges for enhancing the information associated to terms. In fact, these techniques are not incompatible with mathematics. Imagine, for instance that you would like each time a macro \texttt{monoid} is met, to display $M$, you would do for instance:

\begin{verbatim}
\newrobustcmd\monoid{\mathcal M}
\end{verbatim}

Imagine that furthermore, you would like to hyperlink to the definition of a monoid. A standard \texttt{kl} command does the job:\footnote{It is highly advised to not use the quotation notation in math mode or macros.}

\begin{verbatim}
\newrobustcmd\monoid{\kl[monoid]{\mathcal M}}
\end{verbatim}

What is $\monoid$?

would yield:

What is $M$?

The problem 3 But I want more. I want to be able to introduce variables. Even better, I would like to be able to have variables hyperlinking to the place of their introduction, knowing that the same variable name may mean different things depending on the lemma or proof we are in. Hence, I want to properly control the scope of knowledges.

To be done, this requires to use scoping. The principle of scoping is that a knowledge can be attached to a particular context. This is particularly true when typesetting mathematics: a variable is meaningful inside a statement, and inside the proof of the statement. Furthermore, the same variable name may reappear elsewhere with a different meaning.

The following code gives an idea of what is possible using scoping:

\begin{verbatim}
\knowledgeconfigureenvironment{theorem,lemma,proof}{}
[...]
\begin{lemma}\label{theorem:main}
   \knowledge{n}{notion}
   For all number $\intro n$, [...]
\end{lemma}
[...]
Here $\kl n$ is an undefined knowledge.
[...]
\begin{proof}[Proof of theorem\textsuperscript{\ref{theorem:main}}]
   \knowledgeimport{theorem:main}
   Inside the proof, $\kl n$ is hyperlinked to the theorem...
\end{proof}
\end{verbatim}

More on scoping can be found in Section 3.5.

The use of variants of \texttt{kl} is also useful for typesetting mathematics. It allows for instance, to implicitly execute the \texttt{knowledge} command at the same time of the introduction. See 3.4.3 for more detail.
3 Usage of the \texttt{knowledge} package

3.1 Options and configuration

Options are used to activate some capabilities. Some options have to be used when loading the \texttt{knowledge} package, while some others can also be used inside the document thanks to the use of \texttt{\textit{knowledge}configure}. In this section, we review these \textit{package options}.

3.1.1 Options at package loading

The options that can be used in the optional parameter of \texttt{\usepackage} when loading the \texttt{knowledge} package belong to the following classes:

\textbf{Writing mode} The paper, electronic or composition modes are possible (composition is by default) (see Section 3.1.2 for more details). These modes change several default rendering settings.

\textbf{Other packages} some of the options concern the loading and the use of other packages (hyperref, xcolor, ...). This is explained in Section 3.1.3.

\textbf{Configuration options} as used by the command \texttt{\textit{knowledge}configure} can be used when loading the package.

\textbf{Scoping} The scope option makes the package aware at a fine level of the structure of the document (see Section 3.5 for explanations). This provides, for instance, the possibility to define pieces of knowledge that are attached to a section of the documents.

3.1.2 Writing mode

Other package options of \texttt{knowledge} concern the \textit{writing modes}. There are three of them:

- In \texttt{paper} mode, the paper is rendered as for printing: in particular, no informative colors are visible. Hyperlinks are nevertheless present.

- In \texttt{electronic} mode, the document has some colors witnessing the existence of the links for the reader to know that clicking is available.

- In \texttt{composition} mode, the document has colors helping the writing: undefined knowledges appear explicitly, anchor points are displayed, and so on.

Activating the modes is obtained either at load time using one of:

\begin{verbatim}
\usepackage[paper]{knowledge}
\end{verbatim}

or \begin{verbatim}
\usepackage[electronic]{knowledge}
\end{verbatim}

or \begin{verbatim}
\usepackage[composition]{knowledge}
\end{verbatim}

or by setting before loading the variable \texttt{\textit{knowledge}packagemode} as in:

\begin{verbatim}
\def\knowledgepackagemode{paper}
\end{verbatim}

The idea is that this can be used in automatic compilation scripts. For instance,
launching in a terminal:

```
pdflatex "\def\knowledgepackagemode{electronic}\input{file.tex}"`
```

would result in compiling 'file.tex' using knowledge in electronic mode.
The following primitives are available in order to be mode-dependent:

```
\IfKnowledgePaperModeTF{true code}{false code}
\ifKnowledgePaperMode true code [\else false code] \fi
\IfKnowledgeElectronicModeTF{true code}{false code}
\ifKnowledgeElectronicMode true code [\else false code] \fi
\IfKnowledgeCompositionModeTF{true code}{false code}
\ifKnowledgeCompositionMode true code [\else false code] \fi
```

3.1.3 Automatic loading of other packages

A certain number of package options coincide with the loading of other packages. For the moment, the packages that are concerned are hyperref, xcolor, and makeidx.

For activating these functionalities, it is sufficient, either to load the package before the knowledge package, or to name it explicitly as an option for knowledge. Loading separately the package is convenient for setting options for it. For instance, a typical preamble may look like:

```
\documentclass{article}
\usepackage[svgnames]{xcolor}
\usepackage[draft]{hyperref}
\usepackage{makeidx}{knowledge}
```

Such a sequence will activate the knowledge package using the features related to xcolor configured with svgnames option, to hyperref configured with draft option, and to makeidx with its standard configuration.

In fact, the syntax when a package is loaded as an option of knowledge is of the form 'package=choice' in which choice can take the following values:

- **active** The package will be loaded, and all the capabilities that it triggers are activated. This is the implicit meaning when nothing more is specified.
- **inactive** The package is not loaded, and no capabilities are activated (even if it had been loaded previously by another \usepackage command).
- **compatibility** The package is not loaded. The directives it used do not cause any error, but have no effect.
- **auto** If the package was loaded before, then the associated capabilities are activated. This is the default behavior when the package is not named while loading.

Currently, the packages that can be loaded are:
hyperref which activates all the (auto)referencing capabilities.

xcolor which activates coloring commands.

makeidx for handling the index automatically.

3.1.4 Configuring and \knowledgeconfigure

Some part of the configuration can be done outside of the \usepackage command that loads the knowledge package. This is done using the \knowledgeconfigure command:

\knowledgeconfigure{configuration directives}

Note that by default, the configuration directives used by \knowledgeconfigure can be used in the optional parameter of \usepackage when loading the knowledge package, but the converse is not true. Configuration directives consists of a comma separated list of elements that can take the following values:

diagnose line= can be set to true or false. It activates or deactivates the line numbering in the diagnose file. False is the default.

fix hyperref twocolumn triggers a hack that solves a known problem that may occur when hyperref is used in two-columns mode.

notion configures the notion directive which is a refined version of autoref.

protect quotation= is followed by a comma separated list of environments in which the quotation notation will be automatically deactivated (surrounded by braces if more than one item in the list).

protect link and unprotect link starts and ends respectively a zone in which the knowledge package do not create hyperlinks. These can be nested. This is typically useful around, e.g. the table of contents.

quotation activates the quotation notation, which allows to use "...", "...@..." and "...@...@..." instead of \kl commands and "..."n, "%..." and "%...@..." instead of the \intro command.

visible anchor points is an option that makes visible or invisible the anchor points of the \AP and \itemAP commands.

3.2 What is a knowledge?

A knowledge is often informally used in this document. Essentially, it captures what is an elementary concept in the document. Internally, a knowledge is identified by three components:

The knowledge name is a TeX string that has almost no limitation (but being well balanced, and containing no #). It is the text entered by the user for defining and using the knowledge.
The *scope* which is a simple string identifying where the *knowledge* is usable.
The scopes are generated by the system. For instance, internally, each section will be uniquely named ‘section-1’, ‘section-2’, and so on (this is invisible for the user). Each knowledge is primarily valid in exactly one such scope. Knowledges defined in the preamble are given the scope ‘document’. Usually, the user refers to scope using, for instance, labels.

The *namespace* is a simple string that is used for avoiding clashes. It is most of the time simply ‘default’. It is ‘style’ for styles (that are internally as knowledges). It is a possibility available to a developer to, when developing a new set of functionalities, use a different namespace for avoiding clashes of names (for instance if one wants a french and an english set of knowledges that should not conflict, and would use separate sets of macros). Usually, a normal user does not see namespaces.

### 3.3 The \knowledge command and variations

In this section, we describe the main commands that create knowledges. The main one is \knowledge. It can also be used in combination with \knowledgedirective, \knowledgestyle and \knowledgedefault.

#### 3.3.1 General description of the \knowledge command

The key command for introducing knowledges is \knowledge. The syntax is:

\[ \text{\knowledge\{knowledge name\}\{synonyms\}\{directives\}} \]

The *knowledge name* is a string describing the knowledge. It may use any combination of symbols (well balanced with respect to brackets). This string will be used to fetch the knowledge. Note (and this is a standard \TeX behavior) that several consecutive spaces is the same as one or a line feed. The *synonyms* are knowledge names given in a ‘|’ separated list. This is a shorthand for writing ‘link=’ directives. (Note that there is another way to define synonyms using the synonym directive). The *directives* consists of ‘key=value’ statements in a comma separated list. There are many directives. A list can be found in Section 5.3. New ones can be defined using the \knowledgedirective command.

The principle of the \knowledge command is to introduce a new knowledge, ready for being used. However, what it does exactly depends a lot on the situations. First, the directives (a comma separated list of ‘key=value’ commands) are parsed, and from it, the namespace and scope of the knowledge are determined, and it is decided if it will be executed immediately or postponed to the next compilation phase. Only then, either the knowledge is written in the kaux file for execution during the next compilation phase, or it is executed immediately. Finally, synonyms are parsed and linked to the knowledge.
3.3.2 Targeting and the corresponding directives

The \knowledge has to decide what to do when defining something. The basic behaviour is as follows.

- If the \knowledge command is used in the preamble, then the knowledge given as argument is defined immediately (the same effect can be obtained using the now directive), and is accessible in the first compilation phase everywhere in the document (one extra phase is nevertheless required if autoref or ref= directives are used, for the hyperref to do its job, or if scope= is used). This is the simplest way to use \knowledge.

- Import/Export (not implemented) writes a document containing a list of \knowledge commands giving access to its content. This is to be imported by another document.

- Otherwise, the knowledge is written in an external file (the jobname.kaux file), and the knowledge will be really usable in the next compilation phase. This is particularly useful in conjunction with the scope option: the knowledge will have a scope depending on where it is introduced (for instance the document, or a theorem, or a lemma). The same knowledge name can then point to different knowledges depending on where it is used.

The targeting directives refine the above defined behaviour:

scope= When using a directive ‘scope=name’, the scope of the definition can be modified. \knowledge will first check if there is an outer area of this name, that accepts knowledge (only scope environments are subject to this unless \knowledgeconfigureenvironment is used, or the scope package option is used when loading the package). If this is the case, the knowledge will be associated to the corresponding instance. For instance, inside a theorem, by default, the scope is the theorem, but adding the directive ‘scope=section’, the knowledge becomes available in the whole section.

If no scope is found using the above search, a label of the given name is searched for, and if it exists, the corresponding scope is chosen.

export= (not implemented) When using this directive, the knowledge will be (furthermore) written in another file, ready for being used in another document. In particular, the knowledge (in the other document) will point to the present one. The details on how this is supposed to work is to be specified.

namespace= Allows to change the namespace. In itself, this is useless. It has to be used in conjunction with new forms of \kl-like commands.

now requires the knowledge to be defined immediately. This may save one compilation phase. The drawback is that the knowledge cannot be accessed before the \knowledge command that has been introduced. It may help for modularity considerations. (for instance a knowledge is used inside a proof, it has no sense to make it available elsewhere, and it is better style to locally define it).
also now requires the knowledge to be defined immediately as well as delayed to
the next compilation phase. This is in particular how autoreferences should
be handled. See the use of \knowledgenewvariant for more examples.

3.3.3 General directives

We give here the list of display directives that are available without loading any
sub packages. A certain number of Boolean directives are available without any
options. These most of the time are used for typesetting the output. Each
of these can be used as ‘bool=true’ (or shortly just ‘bool’), ‘bool=false’ or
‘bool=default’ (that leaves it in the default state, or the one determined by
surrounding knowledges). The general boolean directives are the following:

emphasize forces the text to be emphasized using ‘\emph’,
italic/up forces/unforces italic (be it in math or text mode),
boldface/md forces/unforces boldface (be it in math or text mode),
underline forces the text to be emphasized using ‘\underline’,
fbox puts a box around the text,
typewriter puts in typewriter font (be it in math or text mode),
ensuretext guarantees that text mode is used (using the ‘\text’ macro, thus in
a way consistent with the surrounding style),
ensuremath guarantees that math mode is used,

mathord, mathop, mathbin, mathrel, mathopen, mathclose, mathpunct yield the
corresponding standard \TeX spacing features in math mode,

mathord for an ordinary mathematical object,
mathop for a large operator (such as $\sum, \Pi, \ldots$),
mathbin for a binary operation (such as $+,-,\otimes,\ldots$),
mathrel for a binary relation (such as $=,\lt,\leq,\ldots$),
mathopen for an opening bracket, parenthesis, \ldots
mathclose for an closing bracket, parenthesis, \ldots
mathpunct for a punctuation symbol.

lowercase puts the content in lowercase,
uppercase puts the content in uppercase,

detokenize detokenizes the content, i.e., instead of executing it provides a string
that displays it (this is useful for commands),

remove space
The non-boolean general directives are the following:

**text**={text} will execute the \LaTeX{} code ‘text’ instead of the key used for calling \kl{}. For instance, \knowledge{latex}{text=\LaTeX{}} will typeset ‘\LaTeX{}’ properly when used. Surrounding braces can be omitted if there are no commas. Be careful when linking to such knowledges, since the substitution of meaning will happen for all the knowledges linking to it.

**link**={knowledge} will continue searching the for linked knowledge. Surrounding braces can be omitted if there are no commas. This directive is often bypassed by the use of the optional argument of \knowledge{} defining synonyms or the synonym directive.

**link scope**={label} will continue searching in the scope identified by the label. Surrounding braces can be omitted if there are no commas. If no directive link= is given, then the same key is searched for. This directive is often bypassed by the use of the optional argument of \knowledge{} defining synonyms or the synonym directive.

**synonym** defines the knowledge as a link to the previously defined knowledge (in fact, the most recently defined that was not using synonym). For instance

\begin{verbatim}
\knowledge{Leslie Lamport}
  \{ref={https://fr.wikipedia.org/wiki/Leslie_Lamport}}
\knowledge{L. Lamport}{synonym}
\knowledge{Lamport}{synonym}
\end{verbatim}

results in the two subsequent knowledge names to point to the first one.

**style**={knowledge style} will adopt the styling option of the knowledge style. Surrounding braces can be omitted if there are no commas.

**wrap**={\token} will execute the macro ‘\token’ with as argument the knowledge text before displaying it. For instance, wrap=\robustdisplay, (where \robustdisplay is a variant of \tl_to_str:n removing the trailing space) is used in this document for typesetting the commands.

### 3.3.4 Knowledge styles and the \knowledgestyle{} command

**Styles** are formatting pieces of information, as for knowledges, but that can be used by other knowledges. In some respect, this is very similar to macro directives (see below), but the difference lies in that styles are dynamically resolved, while macro directives are statically resolved. Styles in particular offer the access to some configuration features of the system. For instance, changing the intro style changes the way the \intro{} command is displayed. See below for some instances.

The central command is \knowledgestyle{}, that has the following syntax:

\begin{verbatim}
\knowledgestyle*{style name}{directives}
\end{verbatim}

The optional star permits to overload an existing style (otherwise, this results in an error). The directives follow the same structure as for a normal \knowledge{} command. When defined, a style can be used in a \knowledge{} command using the directives ‘style=style name’ (it will be used when a \kl{} command calls
for the knowledge) or ‘intro style=style name’ (that will be used by \intro commands).

A certain number of default styles are also offered, that in particular includes warning styles. The list is as follows:
kl is the default style for macros using \kl. It can be modified dynamically using the ‘style=’ directive.
kl unknown and kl unknown cont are the default styles used when an undefined knowledge is met.
intro and intro unknown cont are the default styles used when an undefined knowledge is met.

3.3.5 New directives: the \knowledgedirective command

When defining knowledges, it is often the case that the same sequence of directives are used. Macro directives are here for simplifying this situation (see also \knowledgedefault and \knowledgedirective). This is achieved using the \knowledgedirective directive:

\knowledgedirective{name}[optional parameter]{directives}

After such a command has been issued, ‘name’ becomes a directive usable in \knowledge commands, that amounts to execute the comma separated list ‘directives’. The newly created directive may receive a value, that is accessible as #1 in ‘directives’. The ‘optional parameter’ gives a default value. For instance:
\knowledgedirective{highlight}[brown]{color={#1},emphasize,md}

We shall now see \kl{notion A}, \kl{notion B}, \kl{notion C}, as well as the \kl{important notion D}.

3.3.6 \knowledgedirective versus \knowledgedirective

The two commands \knowledgedirective and \knowledgedirective offer ways to systematize the writing of knowledges. These can seem redundant. This is not the case, and for understanding it, it is necessary to understand a bit the way the \knowledge command works.
In general when a `knowledge` (or `knowledgestyle`) command is found, the directives are parsed and a new internal form of the `knowledge` command is written in the `kaux` file, that will be executed during the next compilation of the document. In this phase, some first operations are performed. For instance, in an `autoref` directive, an internal label name is constructed. Executing a `knowledge macro` is done at this moment.

The postponed command is then executed during the next compilation phase (or immediately if we are in the preamble, or if the `now` directive is used). The execution effectively stores the `knowledge` in the system. This is only at that moment that the `knowledge` becomes available to be used by `k1` and similar commands.

When a `k1` command (or similar) is met, it is ‘executed’, and display informations are considered, and in particular styles are called.

Some consequences of this kind of this are as follows:

- `autoref` directives should not be used in the definition of a style, since this would mean that there would be one anchor point for all the `knowledges` that use this style. This is usually not the kind of behavior that we expect.

- configuring the default displays of the system (such as the `intro style= in particular) has to be done through the style mechanism.

- styles use less memory than macros.

### 3.3.7 Default directives: the `knowledgedefault` command

It may happen that a sequence of consecutive `knowledge` commands have to share the same list of directives. The macro directives can help solving this issue. The `default directives` also go in this direction, using the `knowledgedefault`-command:

```
knowledgedefault*{directives}
```

When such a command is applied, then from that point, all `knowledge` commands will use the given directives as default. This will stop when another `knowledgedefault` command is met or the current group is closed. The optional star does not reset the default directives but simply add new ones.

### 3.4 The `k1` command

#### 3.4.1 The standard syntax

The `k1` command has one of the following syntaxes:

```
k1(optional label)[optional knowledge name]{text}
```

Its meaning is to search for the ‘optional knowledge name’ if present, or for ‘text’ otherwise. How this is exactly performed depends on the presence of the optional label. The search process is as follows:
• if an optional label is given, the knowledge is searched in the corresponding scope.

• otherwise, the stack of visible scope instances is processed through (starting from the innermost) until a knowledge of name ‘knowledge name’ or ‘text’, of namespace ‘default’ and this scope is found.

If the ‘knowledge name/text’ has not been found, the style ‘kl unknown’ (or similar styles, as defined by the unknown style= or unknown style cont=) is used, and the text displayed.

• Otherwise, the knowledge is executed. If it is a link= or synonym defined knowledge, the link is followed, and the process continues.

• Finally, all the definitions involved in the knowledge are processed, following a style= if defined, the knowledge is updated (essentially incrementing the counter of use), and the knowledge is displayed.

This general mechanism is used also by other commands that are variations around \kl such as in particular \intro.

3.4.2 The quotation notation

When activated, the quotation mode activates shorthand notations for the \kl and \intro macros. Possible syntaxes are as follows:

"text" uses the knowledge pointed to by ‘text’. Equivalent to \kl{text}.

"text@knowledge" uses the knowledge pointed to by ‘knowledge to display ‘text’. Equivalent to \kl[knowledge]{text}.

"text@@label" uses the knowledge pointed to by ‘text’ in scope ‘scope’ to display ‘text’. Equivalent to \kl[label]{text}.

"text@knowledge@label" uses the knowledge pointed to by ‘knowledge in scope ‘scope’ to display ‘text’. Equivalent to \kl[knowledge](label){text}.

""text"" introduces the knowledge pointed to by ‘text’. Equivalent to \intro{text}.

""text@knowledge"" introduces the knowledge pointed to by ‘knowledge while displaying ‘text’. Equivalent to \intro[knowledge]{text}.

""text@@label"" introduces the knowledge pointed to by ‘text’ in scope ‘scope’. Equivalent to \intro[label]{text}.

""text@knowledge@@label"" introduces the knowledge pointed to by ‘knowledge in scope ‘scope’ while displaying ‘text’. Equivalent to \intro[knowledge](label){text}.

Activating and deactivating the quotation notation is obtained using:

\knowledgeconfigure{quotation}
and deactivating it is obtained using:

\knowledgeconfigure{quotation=false}.

It can also be activated while loading the package.

It is sometimes the case that some packages do use the quote symbol, usually in some environment (this is the case of the \texttt{tikzcd} environment). The \texttt{knowledge} package can be configured to deactivate always the quotation notation when entering the environment. This is obtained using the configuration option \texttt{protect quotation} followed by a list of environments to be protected:

\knowledgeconfigure{protect quotation={env1,env2,...}}

Note that the braces surrounding the list of environments can be omitted if the list contains only one item.

There are nevertheless some situations in which one would prefer to use the original \texttt{\kl} notation:

- When nesting of \texttt{knowledges} is involved, or the \texttt{knowledge} includes the symbol ",",
- when \texttt{quotation} is deactivated (or not activated) because of a conflict
- in particular, this should be avoided in macros, in particular for the math mode, since these may be used one day or another in a \texttt{tikzcd} or similar environment for instance.

3.4.3 Variants of \texttt{\kl}, \texttt{\knowledgenewvariant}, \texttt{\knowledgevariantmodifi er}

It may happen for several reasons that we may want to define new variants of the \texttt{\kl} macros, that essentially perform the same task, but are configured differently. Typical examples may be:

- several sets of \texttt{knowledges} may intersect but should use different \texttt{namespace},
- some \texttt{knowledges} involve macros and for this reason should be non-expanded even if the \texttt{knowledge} command is not met,
- the \texttt{knowledge} command should be performed automatically,
- activate or desactivate the warnings or messages in the diagnose file.

In fact, several macros in this document are instantiation of this mechanism. This is the case for for instance for \texttt{\intro, \phantomintro, \reintro} or \texttt{\mathkl} etc...

The macro for introducing a new \texttt{variant of \kl} is:

\knowledgenewvariant\variant{variant directives}

and is similar to the one for modifying the behavior of a \texttt{variant of \kl}:

\knowledgevariantset\variant{variant directives} .
These commands define/modify a/the macro \texttt{\textbackslash variant} that uses the same syntax as \texttt{\textbackslash kl}. The \texttt{\textit{variant directives}} consist of a comma separated list of directives as follows:

- **namespace=namespace** declares in which namespace (a string) the knowledge are to be searched. This means in particular that the \texttt{\textbackslash knowledge} concerned should be defined using the the proper \texttt{namespace=} directive.

- **default style=**, **unknown style=**, **unknown style cont={list of style names}** declares the style name to be used (1) by default when the knowledge is found, (2) when it is not found for the first time, and (3) the subsequent times.

- **style directive={directive names list}** defines a list (comma separated) of directives that can be used in a \texttt{\textbackslash knowledge} command to modify the aspect (for instance, the \texttt{\textit{intro}} behavior is modified by the \texttt{\textit{intro style=}} directive, while the \texttt{\textbackslash kl} command is configured using the \texttt{\textit{style=}} directive). If the directives do not exist, these are created.

- **auto knowledge={directives}** declares that the use of \texttt{\textbackslash variant} should automatically execute a \texttt{\textbackslash knowledge} command, and what should be the directives it uses. See examples below.

- **unknown warning=true/false** activates or deactivates the warnings when a knowledge is not found (for instance, these are deactivated in \textit{paper mode}). True by default.

- **unknown diagnose=true/false** activates or deactivates the corresponding messages in the diagnose file. True by default.

- **suggestion={directives}** configures the directives to be suggested in the diagnose file when the knowledge is unknown.

- **PDF string={code}** gives a substitute text for \texttt{\textit{hyperref}} to use for producing the bookmarks. This code has to be expandable. The code may use three parameters; \texttt{\#1} is the main text of the command, \texttt{\#2} is the optional parameter, and \texttt{\#3} is the scope. The macro \texttt{\textbackslash IfNoValueTF} of the package \texttt{xparse} can be used to test if the second and third arguments are present. By default, the code is \texttt{\{\#1\}}. Note that the star syntax cannot be used in this context. It the expected result cannot be achieved using this directive, the less convenient macro \texttt{\textbackslash texorpdfstring} of the \texttt{\textit{hyperref}} should be used.

The second feature is to use modifiers. These correspond to the starred version of the command. For instance, one expects \texttt{\textbackslash intro*\textbackslash kl} to reduce to \texttt{\textbackslash intro}. For this, one has to declare explicitly the reduction using:

\begin{verbatim}
\texttt{\textit{knowledgevariantmodifier} \{starred sequence\} \texttt{\textbackslash variant}}
\end{verbatim}

in which the \texttt{\textit{starred sequence}} is of the form \texttt{\textit{variant1}*\textit{variant2}...*\textit{variantk}}. This sequence is declared to reduce to \texttt{\textbackslash variant}. For instance, \texttt{\textit{knowledgevariantmodifier} \{\textit{intro}*\textit{kl}\} \texttt{\textit{intro}}} declares \texttt{\textit{\textbackslash intro}*\textit{\textbackslash kl}} to reduce to \texttt{\textit{\textbackslash intro}}.
3.4.4 Examples of variants of \kl

The best way for introducing new variants is to look at examples. We provide two
of them now. the first one is the configuration of the \kl and \intro commands
as defined in the package. The second one is the code used in this documentation
for displaying macros, defining the macros \cs and \csintro.

The configuration of \kl and \intro It is also interesting to see this code
since it gives more ideas on how to modify the standard behaviour of these com-
mands correctly.

```
\knowledgestyle{autoref link}{autoref link}
\knowledgestyle{autoref target}{autoref target}
\knowledgestyle{invisible}{invisible}
\knowledgenewvariant\kl{
    namespace=default,
    default style={\kl,autoref link},
    unknown style= \kl unknown,
    unknown style cont= \kl unknown cont,
    style directive= style
}
\knowledgenewvariant\intro{
    namespace= default,
    default style= {\intro,autoref target},
    unknown style= \intro unknown,
    unknown style cont= \intro unknown cont,
    style directive= intro style
}
\knowledgevariantmodifier{\intro*\kl}\intro
```

Note that \reintro and \phantomintro are defined using similar code.

Displaying control sequences The second code example is used in this doc-
ument (the documentation of the package) and consists of two macros \cs and
\csintro which have the following semantics:

- these have the same syntax as \kl and \intro respectively.
- these are used to display control sequences without executing it,
- if \csintro is never used, it appears in black,
- is \csintro is used, then it is in color blue, and the calls to \cs are in
dark blue, and furthermore, the \cs calls possess an hyperlink to the call to
\csintro.
- no \knowledge command is necessary, and no warnings are issued.
Several things can be noted about this code:

- the directives `detokenize` and `remove space` prevent the execution of the argument, and instead display its name, this is important since the argument is a control sequence,
- the directives `typewriter`, `up` and `md` give a uniform aspect (no italic, no boldface) to the result in all contexts,
- the namespace is set to be different from the default one, avoiding possible clashes with `\kl`,
- when a `\csintro` command is met, the corresponding `knowledge` command is automatically issued, in particular with `scope=document` for guaranteeing the visibility of each command everywhere in the document,
- the `also now` directive is necessary for the compilation to (possibly) stabilize in two iterations, since it uses the proper `\label` already at the first iteration (without `also now`, it would be performed on the second one only, and with just `now`, it would be visible only by the uses after the introduction).
- warnings and diagnose information is explicitly eliminated.
3.5 Scoping

Rapidly, when long documents are in construction, one wants knowledges to be isolated in some subparts. For instance, one may want that a temporary definition in a proof to not leak elsewhere in the document where the same term could be used with a different meaning. Some definitions may be only meaningful in, say, the current section/part. This is in particular true when one aims to track single variables: Clearly, a variable named \(x\) can occur in several lemmas, but each of them correspond to a distinct ‘introduction’ location. For handling this situation, the \texttt{knowledge} package possesses some scoping features.

Note that the use of \texttt{scopes} may overconstrain the latex code, in particular if the \texttt{scope} is used. The result is that a code that was compiling before activating the option, will not after doing it. This may happen for instance if one uses a \texttt{section} command inside a a \texttt{theorem} environment. Of course, though this is not forbidden by \LaTeX, this is not something that one usually wants to do. At any rate, the scope handling of the \texttt{knowledge} package — at least in its current form — complains in such a situation. Indeed, it attempts to reconstruct a full hierarchically decomposed version of the document. In the above example, when the \texttt{section} command is met, the package attempts to close the outer section, but this would force the theorem also to be closed. However, closing a theorem requires an \texttt{end} command. Error...

3.5.1 What is the structure of \texttt{scopes} in a document

To start with, one needs to understand what are the possible \texttt{scopes}.

- By default, all the body of the document belongs to a \texttt{scope} called ‘document’. The user can open new scopes using the \texttt{scope} environment:

\begin{verbatim}
\begin{scope}
  \knowledge{local notion}{color=green}
  Here is a \kl{local notion} that appears in green.
\end{scope}
\end{verbatim}

But this \kl{local notion} is undefined.

Note that scoping is independent from the grouping mechanism of \LaTeX. The user can also declare an environment (existing or to exist) to behave like \texttt{scope} using the command \texttt{knowledgeconfigureenvironment}.

- The use of the \texttt{scope configuration option} reconstructs the structure of the document, and \texttt{scopes} will be created for sections, subsections, itemize, items, and so on. But be cautious, this feature, though working, may cause some compiling document to not compile anymore if some weird (and unnatural) nesting of scopes are used.
3.5.2 How is chosen the scope of a knowledge?

In general, when a \texttt{knowledge} command is used, the system tries to figure out what should be its scope:

- If the command occurs in the preamble, then the default scope will be ‘document’.
- Otherwise, the information is searched for in the stack of visible scope instances which means that the knowledge will be defined at the level of the innermost surrounding scope that ‘attracts knowledges’. If the scope option is not activated (and the user did not perform its own configuration), this is the innermost scope environment (or similar environment if \texttt{knowledgeenvironment} has been used), or ‘document’ if the declaration is not in the scope. If the scope option is used, this will be the innermost lemma, proof, or theorem in the context.

- This default behavior can be modified using the \texttt{scope=} directive. The \texttt{scope=} directive can be followed with a scope level, such as ‘section’, ‘subsection’, ‘chapter’ or ‘itemize’ (in particular in combination with the scope option), that will be looked for in the current context and will receive the knowledge. The directive can also be followed by a label name, and the active scope at the moment of this label will be used.

The following code (that requires the scope option for being functional) should be self explanatory:

```latex
\section{First section}
\label{section:first}
\knowledge{one}{scope=section,color=green}
\knowledge{two}{scope=some label,color=green}

\begin{scope}\label{some label}
  Here \texttt{one} and \texttt{two} are defined.
\end{scope}
Here \texttt{one} is defined but \texttt{two} isn’t.

\section{Second section}
Here neither \texttt{one} nor \texttt{two} is defined. However, I can still use them using \texttt{one@section:first} and \texttt{two@some label} (or "one@section:first" and "two@some label", or using the \texttt{knowledgeimport{section:first}}).
```

3.5.3 Accessing other scopes, the \texttt{knowledgeimport} command

Something important is missing so far: one rapidly wants to access to knowledges that do not exist in the current scope. For instance, a notion is used in a section of a document, and one would like to refer to it in the introduction. Another case
is that of a notion or a mathematic variable that is introduced in the statement of a theorem, and should be accessible inside the proof. There are essentially two ways to access such distance knowledges: either use the \kl[label]{text} command (or the equivalent "\ldots\@\ldots" notation), or use the \knowledge-import command. We describe the second possibility now. The syntax is:

\knowledgeimport{label}

The result is that the knowledges in the scope identified by the label are now accessible until the closure of the current scope.

For instance:
\begin{configureenvironment}{theorem,proof}{}
[...]
\begin{theorem}
\label{theorem:1}
\knowledge\alpha{autoref,color=red}
Let $\intro\alpha$ be an integer [...]
\end{theorem}
[...]
Here $\kl\alpha$ is unknown.
[...]
\begin{proof}
\knowledgeimport{theorem:1}
But now $\kl\alpha$ points to its definition.
\end{proof}

3.5.4 Managing scoping environments

The user can also declare an environment to behave like scope using the command \knowledgeconfigureenvironment, as well as adapt some of its characteristics using scope directives.

\knowledgeconfigureenvironment{environments}{scope directives}

For instance:
\knowledgeconfigureenvironment{lemma,theorem,fact,proof}
{knowledge=attracts}
will induce the corresponding environments to have internal knowledges.

Most of the times, it is not necessary to use scope directives.

Remark 1 Note that (in the current implementation) it is necessary to use the commands \begin and \end. Hence \proof...\endproof would not trigger a scoping environment while \begin\proof...\end\proof would.

The scope directives are low level and advanced features. These should not be used in general. Their list is the following:
scope=true/false tells whether an environment should induce a scope. For the moment, this is not used (as soon as configured, it always behave like a scope).

label=none/accepts tells whether a \label command can refer to an instance of this area.

evironment=true/false should be set to true if the scope has to be opened whenever an environment of same name is opened using the \begin and \end commands of \LaTeX.

autoclose=true/false means that the closure is triggered by another event (closure of another enclosing instance, or pushing of an area that requires its closure). It should be true for \LaTeX environments, and false when configuring, e.g. \section to open an scope (since the end of the section is automatic: when another section is opened, or some higher level sectioning command).

parents=\{area1,area2,...\} takes a comma separated list of areas that are allowed as parent. For opening the area, some enclosing instances may be automatically closed for reaching such a parent (if their autoclose= directive is set to true).

push code={code} defines the code to be executed when the area is pushed (each time, these are added).

pop code={code} defines the code to be executed when the area is popped (added too).

occurrences=once/multiple/recursive can be one of ‘once’ if the area can only have one instance in the document, ‘multiple’ if there can be several instances, but not nested, and ‘recursive’, if there is no restriction.

forces=area requires a specific area as an ancestor of this area. This ancestor is implicitly pushed if necessary.

3.6 Error handling

By default, the knowledge package tries to not stop the compilation unless a serious problem has been found. In particular, it is possible to write an entire document using \intro and \kl commands or the quotation notation without ever introducing a knowledge, and only in the end provide this information. This is a feature: as opposed to normal macros, not defining a knowledge should not stop the real work, which is the writing of the document.

Very often there is some undefined knowledge. Such knowledges are displayed using kl unknown and kl unknown cont) styles when issued by \kl; using intro unknown and intro unknown cont when issued by \intro. The detail of the problems are then gathered in the diagnose file.

3.7 The diagnose file

The diagnose file is a file that is created when the knowledge package is used. It enormously eases the use of the package, and it is a good habit to sometimes check it. It gathers a certain number of informations, that can be warning, code
to be used, or simply information. This file has the name of the tex document 
with the extension .diagnose. Its content is divided into clearly identified parts. 
Depending on the used options, some of these parts may appear or disappear.

**Undefined knowledges** in this section are listed all the knowledges that have 
been unsuccessfully searched for. These are given in a line of code of the form 
\knowledge{undefined knowledge}{}. One result of this way of describing 
is that copying the content of this section to the document itself will solve 
all problems of undefined knowledges. It is an efficient way, when one has 
written a document without caring so much about knowledges to copy the 
content of this section, and then modify it/reorganize it, in order to suit ones 
purposes. By default, no suggestion is offered (i.e., the {} braces are empty). 
Suggestion can be automatically configured using the suggestion= directive 
of the macros \knowledgenewvariant and \knowledgesetvariant. 

For instance, using:

```
\knowledgesetvariant\kl{notion}
```

the directive notion is suggested for more directly copying the content.

**Autoref not introduced** This section lists all knowledges that were declared us-
ing the autoref directive (this can be the case indirectly using, e.g. notion), 
but have not been introduced in the document. When a document reaches 
its final states, this section should be empty. Usually, one should add the 
corresponding \intro or \phantomintro command somewhere in the text.

**Autoref introduced twice** In this section, all knowledges that were declared us-
ing the autoref directive and introduced using \intro or \nointro more 
than once are listed. When a document reaches its final states, this section 
should be empty. Consider using \kl or \reintro for solving the problem. 
Note that this may be caused by an \intro used in some title (and repeated 
in the table of contents).

By default, the diagnose file does give the file and the line as a comment before 
every error or warning it lists. This can be cancelled using the diagnose line 
boolean option:

```
\knowledgeconfigure{diagnose line=false}
```

### 3.8 Other packages

#### 3.8.1 The xcolor option

The xcolor option is used if one wants to change colors. It is good to always load 
it since it also triggers coloring for debugging. It triggers colors in the warning 
styles that can be useful in debugging. It also offers two new directives:

- **color=** where in ‘color=name’, name is a color description following the syntax 
of the xcolor package.
colorbox= surrounds the text with a colorbox of given color (following the syntax of the xcolor package).

Loading the package before is necessary for changing the options of the xcolor package (for instance for using svgnames).

3.8.2 The hyperref option

Activating the hyperref option The hyperref option loads the hyperref and triggers a certain number of link-related features. This is done either by the command:

\usepackage[hyperref]{knowledge}

or by loading the hyperref before the knowledge package.

The directives activated by the package are:

- \url= for hyperlinking to an external document
- \ref= for hyperlinking inside document
- \protect link it a boolean for protecting from the creation of nested hyperlinks,
- \autoref for relating objects with their definition
- \autorefhere similar, and used implicitly for math

The package comes also with the configuration directive hyperlinks= which is a boolean deactivates or reactivates the links.

Functionnalities triggered by the hyperref option

- \ref= {label} puts an hyperlink pointing toward a label inside the document (the braces can be omitted when there is no comma).
- \protect link disables the inside hyperlinks,
- \url= {url address} puts an hyperlink to an (external) url (the braces can be omitted when there is no comma).
- \autoref activates the ability to introduce once, use several times an instance. This is very convenient when writing scientific documents with many notions. This is the basic directive activating the features of the \intro command.
- \autorefhere puts immediately a label at the location of the definition, and makes all \kl occurrences of this knowledge hyperlink to this location.

The \autoref directive The \autoref directive is among the most useful offered by the knowledge package. When set, the knowledge should be used with both \intro (exactly once) – or the """"..."""" and """"...@..."""" notations (and variants) if quotation is active – and \kl (possibly several times) – or the "..." notation if quotation is active. The use of \kl will hyperlink to the location of the \intro. The syntax of \intro is the same as for \kl:

\intro[optional knowledge name]{knowledge name}
Hint. Though the \texttt{intro}-command can be used in the title of, e.g. sections, without any errors, this may cause a warning when a \texttt{table of contents} is used: the command is executed twice, once in the \texttt{table of contents}, and once in the document itself.

\begin{document}

In this document, we will see the very important notion of "houses".

Let us define a "house" to be a building that functions as a home.

\end{document}

yields

In this document, we will see the very important notion of houses.

Let us define a house to be a building that functions as a home.

The variant \texttt{intro*} makes the next \texttt{kl} command behave like \texttt{intro}. This is useful in particular in math mode:

\begin{verbatim}
\newcommand\monoid{\texttt{\[\mathcalM\]}}
\begin{document}

Let now $\texttt{\intro*\monoid}$ be a monoid.

Remember now who is $\texttt{\monoid}$.

\end{document}
\end{verbatim}

Hint. This does not work in \texttt{align*} and similar environments. Section 4.5 gives some solutions.

The \texttt{phantomintro} version:

\begin{verbatim}
\phantomintro{}\texttt{knowledge}\{knowledge\}
\end{verbatim}

takes a \texttt{knowledge}, and introduces it at the current location, without displaying anything. This is behaves like an invisible intro, i.e., essentially an abbreviation
for \intro[knowledge]{}. This can be used as a workaround in environment like align* that do not allow the use of labels (see Section 4.5).

The \nointro command:

\nointro{knowledge}

does not display anything and silently prevents the knowledge from issuing warnings because it is not introduced.

The \reintro command:

\reintro[optional knowledge]{knowledge}

is displayed as for \intro, but without being an anchor for hyperlinks, and without counting as a real \intro. It is used if there are for some reason several places that should look like an introduction (typically in the same paragraph), but count as a single target. There is a variant \reintro* that makes the next \kl command behave like a \reintro (similar to \intro* with respect to \intro).

Knowledges that use this directive can be parameterized by modifying the style intro.

For modifying the display of knowledges introduced by \intro, there are new directives:

\intro style= that takes the name of a style as argument. This style will be used when the knowledge is used in a \intro or \reintro command.

\autoref target declares the knowledge to be the target of the autoref (this is implicit when using \kl).

\autoref link requires a link to the target of the autoref to be produced (this is implicit when using \kl).

See the use of \knowledgesetvariant for examples of configuration.

The \autorefhere directive

The \autorefhere directive silently introduces an anchor point at the location of the \knowledge command invoking it. Uses of \kl commands will be hyperlinked to this location.

In some sense, an \autorefhere directive can be understood as the sequence of a \intro directive that would be immediately followed by the corresponding \intro command. This is a bit better since using \autoref in the body of the document requires three phases of compilation (two only if in the preamble). However, the \autorefhere directive does only require two (as for normal labels).

In fact, this \autorefhere directive is what is used underneath when introducing mathematical variables, and should be used for implementing similar behaviors.
Using anchor points  The directives autoref and autorefname use underneath the hyperref package. This means that it puts a label at the place of the \intro command, and then points to it. However, the semantics in this case, is to jump to the beginning of the surrounding ‘region’. If the \intro happens in a ‘section’ (but not inside a theorem-like environment) then the \kl command will point at the beginning of the section, possibly 10 pages above the definition itself.

The standard solution in the hyperref package is to use the \phantomsection. This means defining anchor points in the document that will be the target of hyperlinks.

We offer here new commands for helping using this feature:
\AP declares an anchor point at the left of the current column, at the height of the current line. If the configuration option visible anchor points is set (and this is the case by default), a mark will show the precise location of the target. Be careful: it does not work in some situations, like, for instance inside the optional argument of an \item command (but this is ok elsewhere in an itemize environment), or inside a some macros in mathmode (e.g. fractions). In the particular case of \item, one should use instead:
\itemAP Similar to \AP, but to be used instead of an \item.

Usually putting an \AP (a standard command of the hyperref) at the beginning of every paragraph, and replacing \item by \itemAP in itemize-like environments is most of the time good and safe option.

For instance:
\AP
In order to describe what is a \kl{monoid}, let us us first define a \intro{product} to be an associative binary operator, and a \intro{unit} to be […]
\begin{description}
\itemAP[A \intro{semigroup}] is a set equipped with a \kl{product}.
\itemAP[A \intro{monoid}] is a \kl{semigroup} that has a \kl{unit}.
\end{description}
yields

In order to describe what is a monoid, let us first define a product to be an associative binary operator, and a unit to be […]

A semigroup is a set equipped with a product.

A monoid is a semigroup that has a unit.

One can check that the different knowledges are properly hyperlinked, and that precise targets are the one described by \AP and \itemAP. For helping debugging the anchor points, these are by default made visible as (red) corners on output.
When the `knowledge` package is loaded with the `paper` option this graphical help disappears. This can also be deactivated using:

\knowledgeconfigure{visible anchor points=false}

### 3.8.3 The `makeidx` option

**Activating the `makeidx` option** The `makeidx` option loads the `makeidx` package and triggers a certain number of link-related features. This is done either by the command:

\usepackage[makeidx]{knowledge}

or by loading the `makeidx` before the `knowledge` package.

**Features** When activated, it becomes possible to trigger the \index command when a \kl command is used. The following directives are use:

- **index** = is the version that uses the standard syntax of the \index parameter.
- **index key** = takes as argument the index key: a text that is used for identifying the index entry (usually an accent free version of it).
- **index parent key** = makes the index entry be a subentry of the given main index entry.
- **index style** = makes the index entry be a subentry of the given main index entry.

### 3.9 Dealing with math

This part is under development.

### 3.10 Fixes

In this section, we present some fixes that have been added to help the user solve problems.

**Hyperref and twocolumn** It happens that the `hyperref` and two-column mode yields a fatal error. This happens when a link spans across the boundary between two pages. This is an issue which is not related to the `knowledge` package, but becomes severely more annoying when more links have to be used. A workaround can be tried by using using

\knowledgeconfigure{fix hyperref twocolumn}.

I do not know to which extend it is compatible with various classes...
3.11 Predefined configuration

3.11.1 The notion directive

The configuration option notion is activated using:

\knowledgeconfigure{notion}

It automatically configures a directive notion which is an autoref displayed properly:

- In paper mode, the \intro commands (not in math mode) are emphasized, while the \kl commands are displayed as normal. It has the aspect of a normal paper.

- In electronic mode and composition mode (with the xcolor package), notions are furthermore typeset in blue when introduced, and in dark blue when used. Without the xcolor package, underlining draw the attention to the knowledges (not in math mode).

The behavior of the notion directive is to activate autoref, and to configure the following two styles:

- the style notion is used for normal use,

- the style intro notion is used for introduction.

A typical document using notion could start by the following commands:

\documentclass{article}
\usepackage{xcolor}
\usepackage{hyperref}
\usepackage[paper]{knowledge}
\knowledgeconfigure{notion}
[...]
\knowledge{some text}{notion}

Then the paper is displayer in a colorful way. As soon as the false is replaced by true, the paper becomes black and seriously looking as it should.


4 Some questions and some answers

4.1 How to compile?

As usual with \LaTeX, a certain number of compilation phases are necessary for reaching a document in final form. The problematic point is of course the use of labels, and in particular the \texttt{intro} command. When it is used, and all the \texttt{knowledge} commands are in the preamble, then two phases are necessary. When \texttt{knowledge} commands are used in the body of the documents, then one extra phase is required, meaning three with \texttt{autoref} definitions. This is also the case when \texttt{scoping} is used.

4.2 Problem with \texttt{item} parameters

The use of \texttt{AP} inside \texttt{item} does not work. Do not use \texttt{AP} inside the optional argument of \texttt{item}, and rather use the command \texttt{itemAP}.

Argument of \texttt{k1} has an extra ‘{’}. This is a problem of using optional parameters inside optional parameters such as in \texttt{item[k1[test]{Test}]} . You can surround the content of the optional parameter by two level of curly braces as in \texttt{item[{{k1[test]{Test}}}]} . The notation "⋯" does not have this issue.

4.3 Knowledges and moving arguments (table of contents, ...

The use of \texttt{k1} does not work in (e.g.) the table of content. When the knowledge name contains expandable macros, or accentuated letters, then these are not copied in the table of content as the exact same text, but are expanded/translated. Thus, when the table of content is displayed, the \texttt{k1} command complains of not knowing the knowledge. For instance:5

\begin{verbatim}
\newcommand\Ltwo {\ensuremath{L^2}}
\knowledge{\Ltwo -space}{{\Ltwo -spaces}{autoref}}
\knowledge{étale topology}{{Étale topology}}

begin{document}
tableofcontents
section{On \texttt{\Ltwo -spaces}}
[...]
section{On the \texttt{étale topology}}
[...]
end{document}
\end{verbatim}

5 with \texttt{usepackage[utf8]{inputenc}} and, for instance \texttt{usepackage[T1]{fontenc}} for the accents.
will result in that both knowledges are considered unknown in the table of contents. For the first one, this is due to the expansion of \Ltwo. For the second, this is due to an implicit translation of the accentuated letter into an internal sequence of commands (for instance ‘é’ is translated into the internal sequence ‘\ieC \{\textbackslash e\}’).

Some solutions are as follows:

- Make the macros non-expandable, for instance using \newrobustcmd (of the etoolbox package) or \NewDocumentCommand (of the xparse package, with a different handling of arguments) instead of \newcommand. Hence:

\newrobustcmd\Ltwo {\ensuremath{L^2}}

solves the first problem.

- Using an equivalent text that does not have the problem:

\knowledge{\'etale topology}{link=étale topology}

[...]
\section{On the \kl{\'etale topology}}

- Both problems can be solved using synonyms/links that have no problem. For instance:

\knowledge{Ltwo-space}{link=\Ltwo-space}
\knowledge{etale topology}{link=étale topology}

[...]
\section{On \kl[Ltwo-space]{\Ltwo-spaces}}
\section{On the \kl[etale topology]{étale topology}}

- Other solutions? None so far. I am trying to systematize the treatment of these problems.

Using \intro in a section title causes introducing the knowledge twice. Do not use \intro in titles, but rather \reintro. If you want the section to be the target of the knowledge, then put after the section a \pantomintro command.

\section{On \intro{topology}} \section{On \reintro{topology}} \phantomintro{topology}

Problematic code A solution

4.4 Problems with tikzcd and other issues with the quotation notation

The package tikzcd uses (heavily) the quotes. Thus, it conflicts with the quotation notation. Some other packages may do the same. For solving this issue, the only thing to do is to temporarily deactivate the quotation notation
This can be done either explicitly using before each figure:

\knowledgeconfigure{quotation=false}

and after the figure:

\knowledgeconfigure{quotation}

Another possibility is to force some environment to deactivate systematically the quotation notation when used. For instance

\knowledgeconfigure{protect quotation={tikzcd}}

will deactivate the quotation notation in all the \texttt{tikzcd} environments.

### 4.5 Problems with amsmath

The \texttt{\intro} command does not work in \texttt{align*} or similar environments

It happens that in starred environment (i.e., unnumbered), the package \texttt{amsmath} deactivates the labels. As a consequence the command \texttt{\intro}, which internally uses \texttt{\label} (at least so far), does not work. For the moment, there is no real solution, but a workaround which consists in introducing the knowledge before the incriminated environment using \texttt{\phantomsection}, and then use \texttt{\reintro} inside the environment. Imagine for instance a command \texttt{\SomeCommand}, that inside uses \texttt{\kl[\Somecommand ]}, then:

<table>
<thead>
<tr>
<th>does not work</th>
<th>works</th>
</tr>
</thead>
<tbody>
<tr>
<td>\begin{align*} \intro\SomeCommand \end{align*}</td>
<td>\phantomintro\SomeCommand \begin{align*} \reintro\SomeCommand \end{align*}</td>
</tr>
</tbody>
</table>

### 4.6 Hyperref complains

A fatal error occurs in twocolumn mode. A workaround is to use \texttt{\knowledgeconfigure{fix hyperref twocolumn}}.

### 4.7 Incorrect display

Incorrect breaking at the end of lines (in Arxiv for instance) It may happen that some hyperlinks generated by \texttt{knowledge} are not broken properly at the end of lines. This is an issue with the \texttt{hyperref} package. This in particularly happened for files compiled by the Arxiv system while the file on the local computer was not having any problem.

A workaround is to use the \texttt{breaklinks} option of \texttt{hyperref}. The preamble thus looks like:
4.8 Editor

4.8.1 Emacs editor and quotes

The AucTeX mode in Emacs binds the quote symbol to other characters. This is not convenient when using the knowledge package.

This behavior can be deactivated temporarily using:

\begin{verbatim}
M-x local-unset-key "\n"
\end{verbatim}

or definitively using:

\begin{verbatim}
(defun my-hook () (local-unset-key "\n")
(add-hook 'LaTeX-mode-hook 'my-hook)
\end{verbatim}

4.9 Others

If other kind of problems occur, report them to thomas.colcombet@irif.fr.
5 Resources

5.1 List of commands
\AP introduces an anchor point.
\intro searches for a knowledge and put an anchor to it (to be used with the \autoref directive).
\k1 searches for a knowledge and displays it accordingly.
\knowledge defines new knowledges.
\knowledgeconfigure configures the package.
\knowledgedirective defines a new directive.
\knowledgedefault declares the default directives to be automatically used in \knowledge commands.
\knowledgeimport gives access to knowledges existing in other scopes.
\knowledgegenvariant defines a new variant of \k1.
\knowledgesetvariant configures a variant of \k1.
\knowledgestyle defines a new style.
\knowledgevariantmodifier declares a meaning of * in variants of \k1.
\nointro declares that the knowledge will never be introduced (does not work properly yet).
\phantomintro performs an invisible \intro.
\reintro uses the display style of \intro without introducing an anchor.

5.2 List of environments
export (not implemented) requires exportation of the content.
import (not implemented) declares external resources.
scope Defines a scope in which knowledges are internal.

5.3 List of directives (to use with \knowledge)
autoref Activates the \intro feature (requires the \hyperref).
autoref link activates an hyperlink to the target.
autoref target puts a target for a hyperlink.
autorefhere creates an anchor point that points to the \knowledge command (Requires the \hyperref option).
boldface Displays the knowledge in boldface.
color= Displays the knowledge is the given color (resquires xcolor).
colorbox= Displays the knowledge in a box of the given color (requires xcolor).
detokenize Avoids evaluation of the text.
emphasize Emphasizes the displayed output.
ensuertext Guarantees that the output will be displayed in text mode.
ensuremath Guarantees that the output will be displayed in math mode.
export= (not implemented)
invisible= no display
italic= displays in italic
fbox  Surround the text with a box.
md  Removes boldface typesetting.

notion
index=  Chooses the text to be displayed in the index=.
index key=  the key used to choose the place in the index.
index style=  the style to be used to display in the index.
index parent key=  the parent key in the index.
intro style=  Chooses the typesetting in case of an intro.
italic  Typesets the output in italic.
link=  Follow with the search the linked knowledge.
link scope=  Follow the search in the corresponding scope, using the same key, or the one provided by link= if present.
lowercase  Put all letters of the output in lowercase.
mathord, mathop, mathbin, mathrel, mathopen, mathclose, mathpunct  Selects a spacing behaviour in math mode.
protect link  Disables the hyperlinks inside the link.
ref=  Links to a label inside the document.
scope=  Choose the scope of the definition.
style=  Links to a style.
synonym  Is a synonym of the lastly defined knowledge.
text=  Changes the output text.
remove space  removes the spaces from the input


typewriter  Typeset in as with \texttt.
underline  Underlines the text.
up  Removes italic typesetting.
uppercase  Put all letters of the output in uppercase.
url=  An url to point to (uses the hyperref).

wrap=  A macro used to process the displayed text.

5.4 List of configuration directives (to use with \knowledger-econfure)

composition  switches to composition mode,
diagnose line={true,false}  activates or deactivates the line numbering in the diagnose file.
electronic  switches to electronic mode,
fix hyperref twocolumn  fixes a known problem between hyperref and the two column mode.
hyperlinks={true,false}  activates or deactivates the hyperlinks.
notion  activates the notion directive
quotation={true,false}  activates or deactivates the quotation notation.
paper  switches to paper mode,
protect link and unprotect link  starts and ends respectively a zone in which the knowledge package does not create hyperlinks.
protect quotation={environment list} declares a list of environment in which the quotation notation should be deactivated
visible anchor points={true, false} makes the anchor points either visible or invisible
List of default styles

intro
kl unknown, kl unknown cont
intro unknown, intro unknown cont
notion (if notion is activated)
intro notion (if notion is activated)